Tuning-Up RBF

a report

Alireza Mahmoudian*

1 Network Topology

RBF is short for *Radial-Basis Function* and the reason RBF neural networks are called so, is their hidden neurons' radial-basis activation functions. They are three-layer feed-forward neural networks, with input and output layer like MLPs, and an intermediate layer called RBF layer. RBF layer neurons or RBF neurons for short, are neurons with activation function of the radial from, like *Gaussian function*, giving higher values to vectors near their *center*. The also don't have weighted sum as propagation function as in MLPs. The other neurons activation and propagation functions are exactly like MLP.

From a geometrical point of view, RBF neurons can do what MLP neurons can't, that is discriminating linearly inseparable data with only one layer of hidden neurons. This is because unlike a single perceptron the represent a hyperplane, a single RBF neuron represents a hypersphere.

We can say about RBF exactly same as we said about MLP; It can represent any dataset, but unlike MLP, we have constructive proof to it. It's very simple: At the worst case we can have a single RBF neuron for centered at each training pattern vector. This argument gives us an approach to teaching RBF networks.

2 Learning algorithm

If we follow the weight adjustment approach for learning, we have a very simple job, since all we have is the last layer's input weights that can be trained. We can do it by solving a system of linear equations, which we will solve as finding an inverse matrix M of RBF layer output per each sample multiplied by the matrix of actual training inputs. But since usually our training data is much more than RBF neurons, the matrix M probably won't be square and invertible, and we can't solve the system. So all we can do is to find it's pseudo-inverse, which can approximate the system. So we will have error in classification.

As said, the idea that every single data point could have a RBF neuron of it's own is giving us a clue about a learning algorithm. Of course it is not a good idea to have as many RBF neurons as the data points. Regardless of it's terrible size for a real world database, it would be obviously over-fitting on data it learns, thus loses the most favorite trait of neural networks, that is their ability to *generalize*. So we can keep it as the last resort. But we can start from much less neurons whose centers are not single data points but clusters of where data points are gathered more. Then we can add more neurons for most erroneous data points, if we want to decrease the error, and do this until reach a desired

*SID: 95112095

error level. For initialization of centers, there are many simple clustering techniques we can use.

Using two mentioned methods we can introduce an iterative learning for RBF network, so that starting from an initial network given by a clustering algorithm like *K-means*, at each iteration, the algorithm first calculates the weights matrix first, and then by introducing a new RBF neuron, tries to reduce the error.

Since among the main goals of this report, there is comparison between RBF and MLP, the same benchmarks described in the last report for MLP was used here too.

3 Experiments

As explained in report for MLP, the main we tried to do an exhaustive search for best setting for the meta-parameters, which is called a *tune-up*. Meta-parameters for RBF are less than MLP. We run a function g inside loops which take these parameters at loops: ϵ which is the satisfying error level, σ, that is a parameter of the radial functions that define their *spread* and M which is the number of maximum neurons to add to the RBF layer, thus it also defines the number of iterations. It also takes a parameter r which defines how many times a setting of other parameters should repeat to give an average of the result.

As a classification problem, we then evaluate the result using the accuracy parameter, which was introduced before in the MLP report.

The main program was run with different settings to search for optimal meta-parameters.

4 Remarks

These issues were faced during the experiments and seemed worth to mention here.

- As expected, RBF was so faster than MLP, both in learning and classification phase. One possible reason is the learning algorithm itself, which have at most as many iterations as the number of data points. But we did not have such bound for backpropagation.

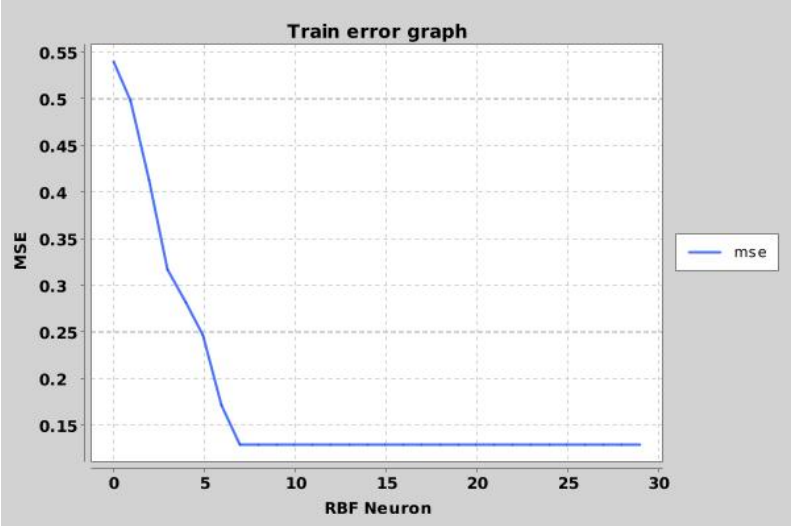
- MLP's multilayer and complex structure can also be responsible for it's low speed. Notice that RBF has the same strength with only one hidden layer.

- We have more confidence in the topology we found for the RBF to be the optimal topology. MLP's topology is more complex, so we would have to search a much vaster space for an optimal topology. Back then, we just blindly tried MLP with different settings and hoped to find it out of luck. But here we are building RBF in a bottom-up manner, making sure that in each step we are giving it more strong.

- An interesting phenomenon happened few times. In some experiments, while the learning algorithm was adding more RBF neurons to the network, at some point the error would not decrease anymore then and stood still. It seemed like the already added neurons have covered the data set and adding more neurons won't help. You can see such condition in figure 1.

- We used here a parameter that was proposed at the class, to evaluate the ratio of accuracy to the number of neurons used, so it will help us to see if we can trade some accuracy with simplifying the network. It is shown here by A/n is in fact

Figure 1: Error per epoch graph in batch mode



calculated as $\frac{Accuracy \times 100}{n}$. But it gives us many results with poor accuracy, just because they have few RBF neurons. It seems we need to give the accuracy term more value or define a lower bound for it.

- The source code for this project, including this report, can be found at `https://github.com/BelegCuthalion/mlp-rbf-tuner`

5 Results

Tables that are shown here are the best results filtered out of nearly 20 thousand attempts to tune up RBF for two mentioned benchmarks. We sort the data by two parameters and picked up first rows from both. The first parameter is accuracy, so we can compare our result with that of MLP. For the results of the first benchmark, we got better accuracy results than MLP, although it's still not satisfying. For the second benchmark, in many runs the network even classified 100% of the testing data. So we can confirm our conclusion in the last report, about neural networks being too much for this benchmark. Two latter tables are top-most results, sorted by *accuracy per neurons* parameter. They've been brought here just to show this proposed parameter needs modification, as mentioned in the previous section, to become a helpful criterion.

Table 1: Benchmark 1: Best results by accuracy

RBF Neurons	ϵ	σ	Last Error	Accuracy	A/n
75.00	0.00	1.90	0.65	0.74	0.99
75.00	0.05	1.13	0.64	0.74	0.99
75.00	0.05	1.45	0.67	0.74	0.99
48.00	0.40	0.01	0.39	0.71	1.48
75.00	0.00	0.16	0.09	0.71	0.95
75.00	0.00	0.50	0.62	0.71	0.95
75.00	0.00	1.29	0.63	0.71	0.95
75.00	0.05	0.48	0.66	0.71	0.95
75.00	0.05	0.52	0.48	0.71	0.95
75.00	0.05	0.54	0.58	0.71	0.95
75.00	0.05	0.55	0.54	0.71	0.95
75.00	0.05	1.50	0.64	0.71	0.95
75.00	0.05	1.78	0.59	0.71	0.95
13.00	0.60	0.31	0.59	0.68	5.21
69.00	0.05	0.07	0.05	0.68	0.98
75.00	0.00	0.37	0.72	0.68	0.90
75.00	0.00	0.57	0.68	0.68	0.90

Table 2: Benchmark 2: Best results by accuracy

RBF Neurons	ϵ	σ	Last Error	Accuracy	A/n
13.00	0.08	0.97	0.07	1.00	7.69
18.00	0.06	0.97	0.06	1.00	5.56
20.00	0.07	0.73	0.06	1.00	5.00
20.00	0.08	0.77	0.07	1.00	5.00
20.00	0.10	0.70	0.10	1.00	5.00
21.00	0.09	0.76	0.08	1.00	4.76
22.00	0.07	0.71	0.06	1.00	4.55
22.00	0.09	0.72	0.08	1.00	4.55
23.00	0.05	0.76	0.04	1.00	4.35
23.00	0.09	0.64	0.09	1.00	4.35
23.00	0.10	0.64	0.09	1.00	4.35
23.00	0.10	0.66	0.09	1.00	4.35
24.00	0.05	0.74	0.05	1.00	4.17
24.00	0.08	0.66	0.08	1.00	4.17
24.00	0.10	0.67	0.10	1.00	4.17
25.00	0.04	0.79	0.04	1.00	4.00
26.00	0.08	0.52	0.08	1.00	3.85

Table 3: Benchmark 1: Best results by ratio of accuracy to the RBF neurons' count

RBF Neurons	ϵ	σ	Last Error	Accuracy	A/n
2.00	0.05	0.01	1.00	0.48	24.19
3.00	0.80	0.91	0.63	0.61	20.43
3.00	0.80	0.71	0.66	0.61	20.43
3.00	0.70	0.61	0.70	0.58	19.35
3.00	0.90	0.51	0.70	0.58	19.35
3.00	1.00	0.31	0.84	0.58	19.35
3.00	1.00	0.61	0.69	0.58	19.35
3.00	0.80	0.41	0.75	0.58	19.35
3.00	1.00	0.81	0.65	0.55	18.28
3.00	1.00	0.81	0.66	0.55	18.28
3.00	0.70	0.61	0.66	0.55	18.28
3.00	0.70	0.91	0.66	0.55	18.28
3.00	0.70	0.71	0.66	0.52	17.20
3.00	0.80	0.81	0.67	0.52	17.20
3.00	0.90	0.21	0.86	0.52	17.20
3.00	0.90	0.41	0.75	0.52	17.20
3.00	0.90	0.71	0.64	0.52	17.20

Table 4: Benchmark 2: Best results by ratio of accuracy to the RBF neurons' number

RBF Neurons	ϵ	σ	Last Error	Accuracy	A/n
2.00	0.02	0.02	1.00	0.23	11.67
2.00	0.00	0.01	1.00	0.20	10.00
2.00	0.05	0.01	1.00	0.20	10.00
11.00	0.02	0.99	0.02	0.93	8.44
11.00	0.03	0.96	0.03	0.93	8.44
2.00	0.00	0.01	1.00	0.17	8.33
2.00	0.00	0.01	1.00	0.17	8.33
2.00	0.00	0.01	1.00	0.17	8.33
2.00	0.00	0.01	1.00	0.17	8.33
2.00	0.01	0.01	1.00	0.17	8.33
2.00	0.01	0.01	1.00	0.17	8.33
2.00	0.01	0.01	1.00	0.17	8.33
2.00	0.01	0.01	1.00	0.17	8.33
2.00	0.02	0.01	1.00	0.17	8.33
2.00	0.05	0.02	1.00	0.17	8.33
2.00	0.08	0.02	1.00	0.17	8.33
2.00	0.10	0.02	1.00	0.17	8.33